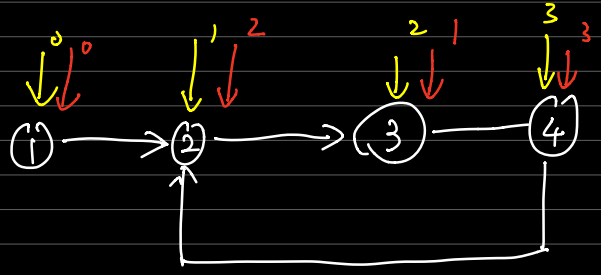
这是一个有用的算法，原地（O(1)）且时间算法度为O(n)用于在 linked的图式线性结构，例如链表中判定是否存在环。此算法称为 Floyd 判圈或者 龟兔赛跑

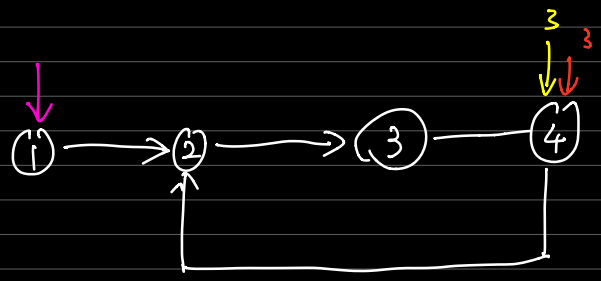
其有两种作用

1、判定是否存在圈

2、判定 圈的入口



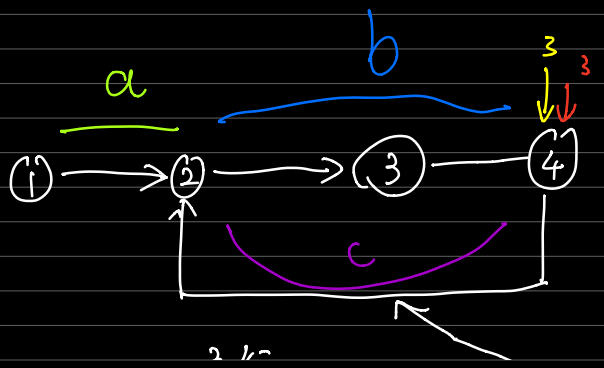
如上图为指针的移动顺序，设指针1，1次移动1位，指针2，1次移动两位，那么，如果存在圈，则两者一定会相遇，相当于多一指针相对多走了若干距离



遇

如上相遇时，从起点另行走一指针，那么其和指针1会在入口相

证明如下：



意味着指针走了

如上三个距离，当 指针 和 指针 相遇时，有

$$slow = (a+b)$$

$$fast = a + n(b+c) + b$$

因此 $2(a+b) = a + n(b+c) + b$

$$a = n(b+c) - b$$

$$= (n+1)(b+c) + c$$

$(n+1)(b+c)$：走了 n 圈

$+c$：的走个 c

a 的距离另外两指针就会走 n+1 圈，再多走 c，刚刚好会落在相遇点，即走了 c 的距离

此种方法可用在其他变种上

#287 寻找重复数：给定一个 n+1 个整数的数组，其数字在 1～n 之间，只有一重复的数，找出这个数

nums = [1, 3, 4, 2, 2]　　ans => 2

nums = [3, 1, 3, 4, 2]　　ans => 3

两种思路：[sort(nums) → search]

另一种思路则是利用图